
華中師範大學
CENTRAL CHINA NORMAL UNIVERSITY
基于PyTorch的CIFAR10分类的分析(基于VGG19网络)
CPU、GPU、多GPU性能分析
汇报人：张国伟（ 2021123388 ）

目录

01 背景介绍

任务描述

基于PyTorch使用VGG19网络在CIFAR-10上训练出一个分类模型

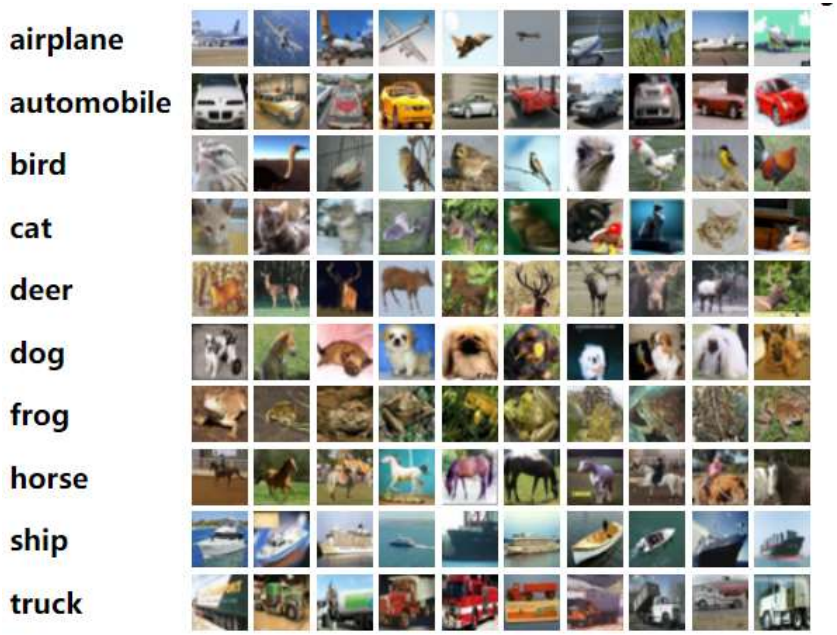
网络模型介绍

VGG-19结构如右图E列，总共有19层，包含
16个卷积层和3个全连接层
参数多，计算量大

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

CIFAR-10数据集介绍

- 10个类别
- 每个类别6000张图片
- 总计有6000张图片
- 训练集50000张，测试集10000张



02 设备参数

01 CPU

CPU相关参数
16 Intel(R) Core(TM) i9-9900K CPU @ 3.60GHz

自己的服务器

CPU相关参数
12 Intel(R) Xeon(R) E5-2678 CPU @ 2.50GHz

租的云服务器

02 GPU

GPU相关参数
GeForce RTX 2080 Ti

本机和云服务器一致

03 实验过程与结果分析

在本地CPU

Batch_size: 128

Epoch: 200

Total_time: 46H22M

Per_Epoch:13M

Accuracy：93.58%

```
Epoch: 197
 [============================================================ 391/391 : 1s292ms | Tot: 13m19s | Loss: 0.001 | Acc: 99.972% (49986/50000))
 [============================================================ 100/100 p: 341ms | Tot: 31s774ms | Loss: 0.302 | Acc: 93.580% (9358/10000)

Epoch: 198
 [============================================================ 391/391 : 1s332ms | Tot: 13m22s | Loss: 0.001 | Acc: 99.986% (49993/50000)0)
 [============================================================ 100/100 ep: 322ms | Tot: 32s64ms | Loss: 0.305 | Acc: 93.510% (9351/10000)

Epoch: 199
 [============================================================ 391/391 : 1s370ms | Tot: 13m19s | Loss: 0.001 | Acc: 99.976% (49988/50000)))
 [============================================================ 100/100 ep: 335ms | Tot: 33s34ms | Loss: 0.301 | Acc: 93.580% (9358/10000)
total_Time consumed:166921.58s
```

处理比较慢，14Min/Epoch

CPU不善于计算

在本地GPU

Batch_size【200Epoch】	time	GPU使用(mem:)11019
128	2671/44M31S(11s)	3179/89%
256	2366/39M26S(10s)	4677/93%
512	2192/36M32S(9s)	4147/95%
1024	2162/36M2S(8s)	5833/97%
2048	2199/**36M39**S(8s)	10105/98%

使用DP 数据并行

本地GPU

Batch_size【100Epoch】	time		GPU使用(mem:)11019
256	1536s/25M36S(13s)	51M12S	3895/94%
512	1437/23M57S(12s)	47M54	6509/97%
1024	1426/23M46(11s)	47M32S	7711/98%
2048	2717/45M17S(23s)	90M34S	7027/99%

不使用DP 数据并行

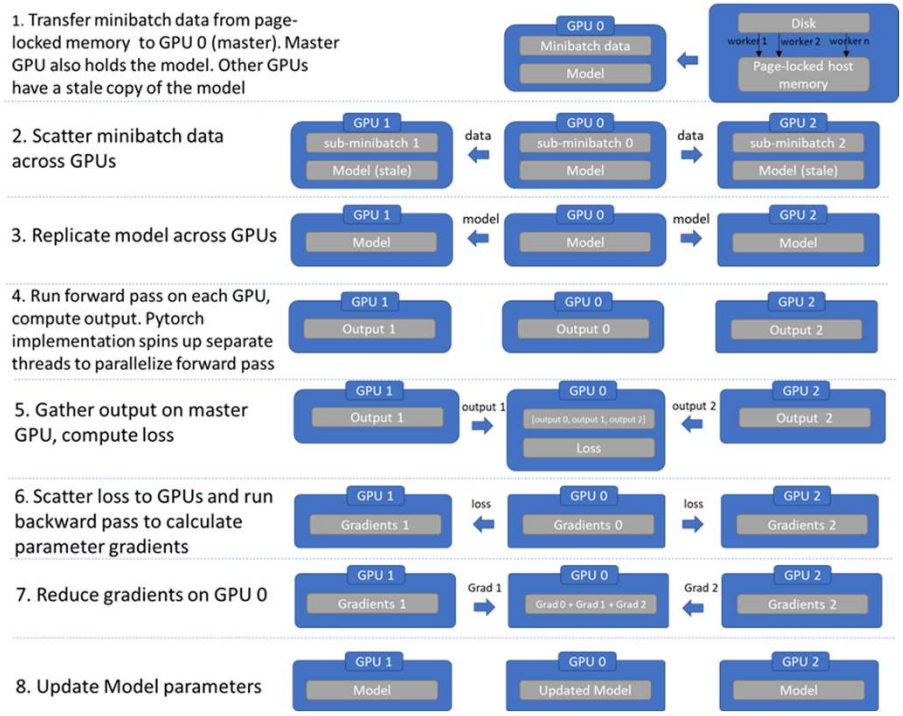
- CPU与GPU相比较，速度提升很大
- 单机使用DP也可提高训练速度，而GPU的使用率低1%
- Batch_size对训练时间的影响，适当提高训练的Batch_size有助于提高训练速度，但不能一直提高
  【batch太大，数据太多，数据加载变慢】
- 之后提升不大，原因：I/O
- Batch_size与GPU利用率的关系

使用本地GPU，GPU的利用率能够达到98%，机器整体的性能是比较好的

Data Parallel

One GPU (0) acts as the master GPU and coordinates data transfer.

Implemented in PyTorch data_parallel module



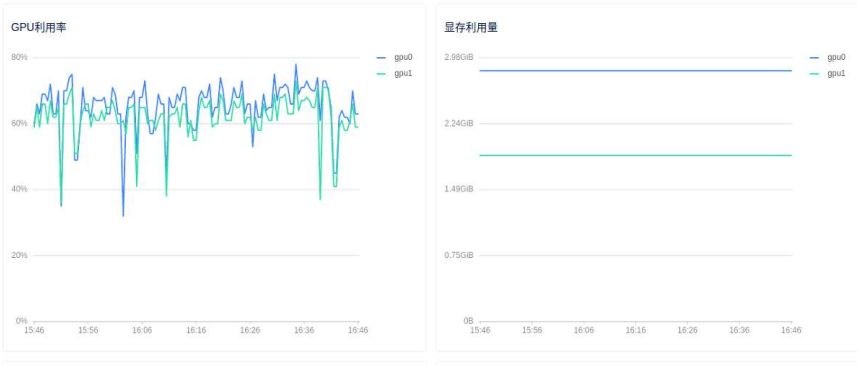
将输入**一个 batch 的数据均分成多份**，分别送到**对应的 GPU 进行计算。**与 Module 相关的所有数据也都会以浅复制的方式复制多份。每个 GPU 在单独的线程上将针对各自的输入数据独立并行地进行 forward 计算。然后在**主GPU上收集网络输出**，并通过将网络输出与批次中每个元素的真实数据标签进行比较来计算损失函数值。接下来，**损失值分散给各个GPU，每个GPU进行反向传播以计算梯度。**最后，在**主GPU上归约梯度、进行梯度下降，并更新主GPU上的模型参数。**由于模型参数仅在主GPU上更新，而其他从属GPU此时并不是同步更新的，所以**需要将更新后的模型参数复制到剩余的从属 GPU** 中，以此来实现并行。DataParallel会将定义的网络模型参数默认放在GPU 0上，所以dataparallel实质是可以看做把训练参数从GPU拷贝到其他的GPU同时训练，这样会导致内存和GPU使用率出现很严重的负载不均衡现象，即GPU 0的使用内存和使用率会大大超出其他显卡的使用内存，因为在这里GPU0作为master来进行梯度的汇总和模型的更新，再将计算任务下发给其他GPU，所以他的内存和使用率会比其他的高。

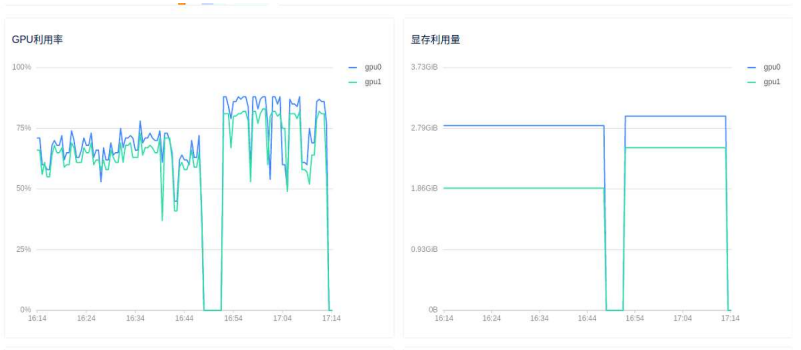
单机双GPU

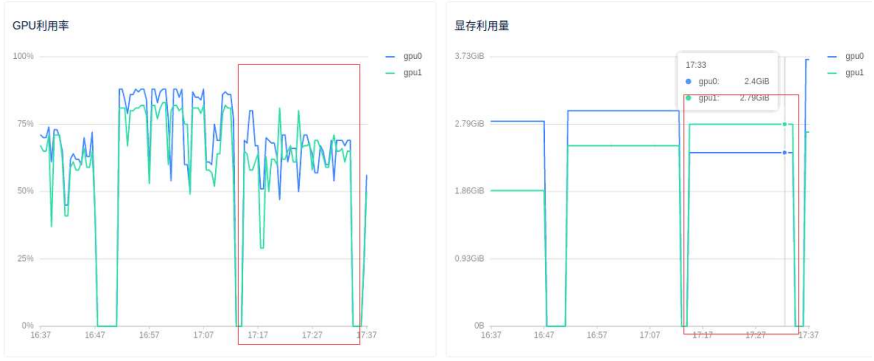
Batch_size【100Epoch】	time	GPU使用(mem:)11019	Time[单机单GPU 本地]	Time【单机单GPU 云】
128(200Epoch)	1H22min（23s）	2.84G/1.88G(74%/67%)		
128	44min			
256	1266s/21min6s(12S)	2.98/1.88(88%/81%)	51M12S	
512	1183s/**19min43s(9s)**	2.4/2.79(50%/80%)	47M54	1163S/**19M23S**
1024	1186/19min46s(9s)	3.69/2.68(79%/80%)	47M32S	90%-92% 3.89G

使用DP 数据并行

实验过程与结果分析



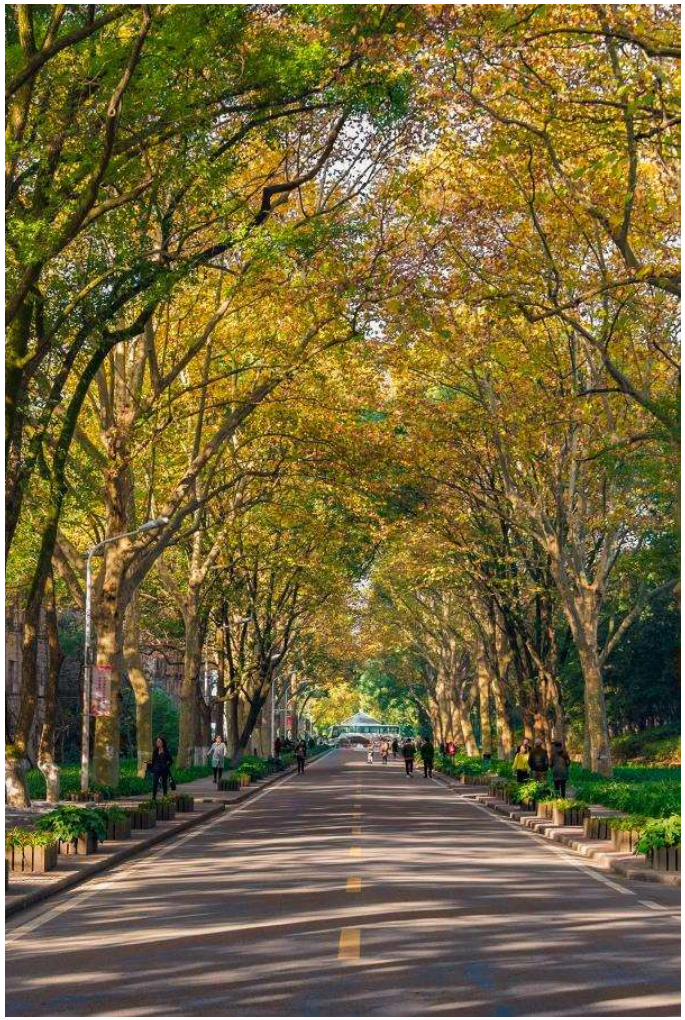




128 | 256
512 | 1024

- 同样是RTX 2080Ti,GPU的利用率要低很多（80%），不及本地最低值，这个限制是由于CPU的性能
- 两块GPU的利用率、显存利用量不均等的，即负载不均衡
- 两块GPU的时间 与 一块GPU的时间 【数据集大小的原因，换成ImageNet 两块GPU的优势凸显出来】
- 怎么提升GPU的利用率，配置与之相匹配的硬件，充分发挥GPU本身的性能

04 总结



- CPU<GPU<多GPU
- 要寻找合适的batch_size,提升训练效率
- GPU里利用率不仅受限于GPU的性能，也受机器I/O与数据集大小的约束。硬件之间要互相匹配
- 使用DP/DPP有助于提升训练速度

華中師範大學
CENTRAL CHINA NORMAL UNIVERSITY

谢谢大家